Set Manipulation with Boolean Functional Vectors for Symbolic Reachability Analysis

Amit Goel
Department of ECE,
Carnegie Mellon University, PA. 15213. USA.
agoel@ece.cmu.edu

Randal E. Bryant
Computer Science Department,
Carnegie Mellon University, PA. 1513. USA.
Randal.Bryant@cs.cmu.edu

Abstract

Symbolic techniques usually use characteristic functions for representing sets of states. Boolean functional vectors provide an alternate set representation which is suitable for symbolic simulation. Their use in symbolic reachability analysis and model checking is limited, however, by the lack of algorithms for performing set operations. We present algorithms for set union, intersection and quantification that work with a canonical Boolean functional vector representation and show how this enables efficient symbolic simulation based reachability analysis. Our experimental results for reachability analysis indicate that the Boolean functional vector representation is often more compact than the corresponding characteristic function, thus giving significant performance improvements on some benchmarks.

1. Introduction

Binary Decision Diagrams(BDDs) [3] enabled the efficient representation and manipulation of boolean functions. Symbolic techniques use BDDs to encode and manipulate sets of states for automatic verification techniques based on state-traversal, such as symbolic model checking. The most common encoding used for state-sets is the characteristic function representation. Boolean functional vectors provide an alternate, though not as widely used, representation.

Boolean functional vectors represent a bit-level decomposition of the state-set which is suitable for symbolic simulation. The decomposition is often more compact than the equivalent characteristic function representation. However, while there are straight-forward algorithms for set manipulations with characteristic functions, similar algorithms do not exist for Boolean functional vectors.

In an early paper on symbolic state traversal, Coudert, Berthet and Madre [6] used symbolic simulation for the image-computation step in reachability analysis, as shown in Figure 1. Starting with the set of initial states, they iteratively perform image computation on a given circuit model. The union of the obtained image and the previously reached states gives the set of reached states for that iteration. The procedure stops (Loop Control) when no new states are discovered, i.e., a fix-point is reached. A selection heuristic is used to choose the smaller of the newly discovered states and all the discovered states for the next iteration.

The image computation was performed by symbolic simulation with Boolean functional vectors. However, all set-manipulation was done with characteristic functions. The conversion between the two representations is costly and since it creates the characteristic function anyway, there are no benefits to using Boolean functional vectors.

In [7], Coudert and Madre replaced the symbolic simulation with a range computation by constraining the transition functions with the characteristic function. This avoids the conversion from characteristic functions to Boolean function vectors. To convert the Boolean functional vector (obtained from the range computation) to a characteristic function, they proposed two algorithms based on recursively splitting the vector. While most state-traversal programs today use the *transition relation* approach with partitioned transition relations and early quantification [8], the *transition function* approach of recursive splitting is still used sometimes, e.g., in [11] where a hybrid approach is used effectively. However, in almost all reachability analysis, the characteristic function is used as the primary set representation. McMillan's conjunctive decomposition [10] is one approach where this is not the case. As we show in Section 2.7, this decomposition is closely related to the Boolean functional vector representation.

Boolean functional vectors are also used in Symbolic Trajectory Evaluation (STE)[4] which is a symbolic simulation based model checking technique. However, the specification language is restricted and does not require fix-point computations, thus avoiding the need for set manipulations.

In this paper we present algorithms for set union, intersection and quantification that work directly on a canonical

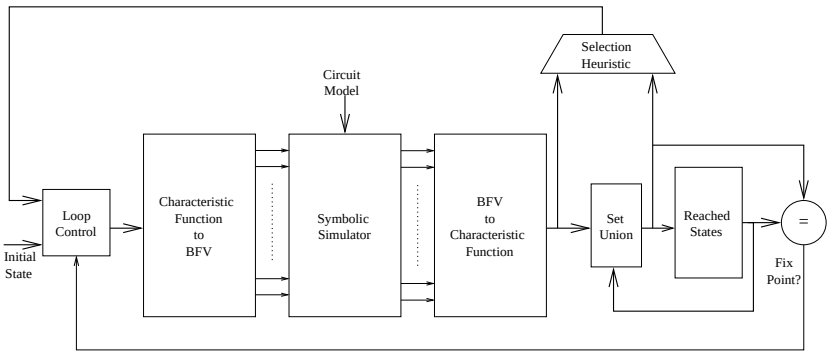

Figure 1. Reachability Analysis with the Coudert, Berthet, Madre [6] approach. State-sets are represented by their characteristic functions for set manipulation and Boolean functional vectors (BFVs) for image-computation by symbolic simulation.

Boolean functional vector representation (we do not have a negation algorithm). These algorithms do not construct the characteristic function either explicitly or implicitly, thus enabling a symbolic simulation based state-traversal using Boolean functional vectors without paying the price for unnecessary conversions.

2 Set Representation and Manipulation

A characteristic function represents a constraint which must be satisfied for a vector to be in the encoded set. Consider the set of vectors $S = \{000, 001, 010, 011, 100, 101\}$ (we abbreviate bit-vectors with bit-strings, e.g., 000 for $[0,0,0]$). In our example, if we use variables v_1, v_2 and v_3 for the first, second and third bits respectively, the characteristic function is $\chi_S = \neg v_1 + \neg v_2$ (Table 1). This expresses the constraint that the first two bits cannot both be 1.

Formally, given variables $V = (v_0, \ldots, v_n)$ and $\mathcal{B} = \{0,1\}$, a *characteristic function* $\chi_S(V)$ represents the set:

$$S = \{X \in \mathcal{B}^n | \chi_S(X) = 1\}$$

Boolean functional vectors, on the other hand, map a given input vector to a vector in the encoded set. For example, the set $S = \{000, 001, 010, 011, 100, 101\}$ can be represented by the Boolean functional vector $F = [v_1, \neg v_1 v_2, v_3]$ as shown in Table 1.

In general, a Boolean functional vector represents the set of Boolean vectors given by its range. Formally, given variables $V = (v_1, \ldots, v_m)$ and $\mathcal{B} = \{0,1\}$, the *Boolean functional vector* $F = [f_1(V), \ldots, f_n(V)]$ represents the set:

$$Range(F) = \{X \in \mathcal{B}^n | \exists Y \in \mathcal{B}^m . F(Y) = X\}$$

2.1 Canonical Representation

Unlike the characteristic function, the Boolean functional vector representation of a set is not unique. The vectors $[v_1, \neg v_1 v_2, v_3]$ and $[v_1 \neg v_2, v_2, v_3 + v_4]$ both represent the set $\{000, 001, 010, 011, 100, 101\}$. Also, there is no Boolean functional vector for the empty set.

v_1	v_2	v_3	χ_S	F
0	0	0	1	000
0	0	1	1	001
0	1	0	1	010
0	1	1	1	011
1	0	0	1	100
1	0	1	1	101
1	1	0	0	100
1	1	1	0	101

Table 1. Representing a set by its characteristic function and a Boolean functional vector

However, for non-empty sets, it is possible to obtain a canonical Boolean functional vector representation by placing some restrictions [6, 13]. The empty set can be treated as a special case.

Firstly, we use exactly the same number of variables as the number of vector components. Each variable corresponds to a choice for one component. The functional vector then represents a mapping from the set of all n length vectors onto the set being represented.

The second restriction placed is that a vector which is in the set must be mapped to itself. Thirdly, we use a distance metric to map vectors not in the set to the nearest vector in the set. For vectors $X = [x_1, \ldots, x_n]$ and $Y = [y_1, \ldots, y_n]$ the distance is defined by $d(X,Y) = \sum_{i=1}^{n} 2^{n-i}(x_i \oplus y_i)$.

The distance metric ensures uniqueness, i.e., no two vectors are equi-distant from a given vector. Note that we have assigned decreasing weights to the bits, starting with the first bit. In general, a different ordering could be used to assign weights. This corresponds to a permutation of the bits. We refer to this order as the *component order*.

For the set $S = \{000, 001, 010, 011, 100, 101\}$, if we use the choice variables v_1, v_2 and v_3 for the first, second and third bits (vector components) respectively, we get the canonical symbolic encoding $F = [v_1, \neg v_1 v_2, v_3]$, shown in Table 1.

Given a characteristic function for a set, the corresponding Boolean functional vector can be obtained by using the conversion algorithms of Coudert, Berthet and Madre [6] or the parameterization procedure of [1]. In our usage, however, we start with the canonical vectors for elementary sets and build other vectors by the set manipulation algorithms discussed below. The distance metric is never used explicitly, but the nearest distance property is maintained by the algorithms.

2.2 Interpreting Boolean Functional Vectors

In developing the set manipulation algorithms for Boolean functional vectors, we found it useful to interpret a Boolean functional vector as an ordered selection process, starting with the highest weighted component.

For $S = \{000, 001, 010, 011, 100, 101\}$, we can choose the first bit to be either 0 or 1. We use the choice variable v_1 to represent this *free-choice*. The value of the second bit is restricted by our selection of the first bit. When the first bit is chosen to be 0, the second bit can be either 0 or 1. However, if the first bit is chosen to be 1, the second bit is forced to be 0. This dependency is captured by the second component function $\neg v_1 v_2$. The third bit value, which can be chosen independent of the first two choices, is represented by the choice variable v_3.

In general, the i-th component will depend on the first i variables only. The i-th component can be represented as $f_i = f_i^1 + f_i^c v_i$ where v_i is the i-th choice variable and f_i^1 and f_i^c are functions of the first $(i-1)$ variables. The function f_i^1 represents the condition under which f_i is *forced-to-one* because of previous selection choices. f_i^c represent the condition under which we have a *free-choice* for f_i. The *forced-to-zero* (f_i^0) condition does not appear in f_i but is easily computed since the three conditions are mutually exclusive and complete. Any two of the three functions f_i^1, f_i^0 and f_i^c are sufficient to define the i-th component.

2.3 Set Union

With the selection interpretation of Boolean functional vectors, a naive union algorithm suggests itself. In selecting a vector from the union, we can choose from either of the operand sets. Hence, the i-th component is forced to a value in the union only when it is forced to that value in both sets. If we have a free-choice in either set, or if one set allows us to choose 1 and the other 0, we have a free-choice in the union for that component.

Consider computing the union S of the sets $S_0 = \{000\}$ and $S_1 = \{011\}$, represented by the functional vectors $[\mathbf{0}, \mathbf{0}, \mathbf{0}]$ and $[\mathbf{0}, \mathbf{1}, \mathbf{1}]$ respectively. Since the first component is forced-to-zero in both sets, it is forced-to-zero in the union as well. For the second component, we get a free-choice since one set allows 0 while the other allows 1. Similarly, by our naive algorithm we would get a free-choice for the third component to give us the Boolean functional vector $[\mathbf{0}, v_2, v_3]$ corresponding to the set $\{000, 001, 010, 011\}$, an over-approximation of the correct union.

The problem occurs because after we make a choice for the second bit in our example, we restrict ourselves to one of the two sets and, hence, we do not have a free choice for the third bit which is forced to one or zero depending on the choice made with v_2.

In order to compute the Boolean functional vector H corresponding to the union of (the sets represented by) F and G, we compute *exclusion conditions* F^x and G^x. Initially, neither set is excluded.

$$\begin{aligned} f_1^x &= \mathbf{0} \\ g_1^x &= \mathbf{0} \end{aligned}$$

Whenever we do make a choice that restricts us to one of the sets, we update the exclusion conditions to restrict the selection procedure to the remaining set. A set is excluded while selecting the $i+1$ component either if it has already been excluded earlier or its i-th component is forced to a value and we selected the other value:

$$\begin{aligned} f_{i+1}^x &= f_i^x + f_i^0 \cdot h_i + f_i^1 \cdot \neg h_i \\ g_{i+1}^x &= g_i^x + g_i^0 \cdot h_i + g_i^1 \cdot \neg h_i \end{aligned}$$

The union can now be computed as:

$$\begin{aligned} h_i^1 &= f_i^1 \cdot g_i^1 + f_i^1 \cdot g_i^x + f_i^x \cdot g_i^1 \\ h_i^0 &= f_i^0 \cdot g_i^0 + f_i^0 \cdot g_i^x + f_i^x \cdot g_i^0 \end{aligned}$$

The i-th bit is forced-to-one in the union if it is forced-to-one in both the sets or if one set is excluded and the bit is forced-to-one in the other set. The forced-to-zero computation is similar.

2.4 Set Intersection

Consider the sets $S_0 = \{000, 010\}$ and $S_1 = \{001, 010, 011\}$ represented by the vectors $F = [\mathbf{0}, v_2, \mathbf{0}]$ and $G = [\mathbf{0}, v_2, \neg v_2 + v_3]$ respectively. While selecting a vector for the intersection, we cannot choose the second bit to be 0 since that would give conflicting values for the third bit.

A conflict is introduced when a bit is forced-to-one in one set and forced-to-zero in the other. To handle these conflicts, we introduce *elimination conditions* E. The function e_i represents the conditions which lead to a conflict downstream, irrespective of the remaining choices. Since there

are no downstream components for the n-th bit, it's elimination condition e_n is 0. The elimination condition e_{i-1} includes conflicting choices for the i-th bit position and also further downstream conflicts which cannot be resolved by either value of the i-th choice variable v_i:

$$\begin{aligned} e_n &= \mathbf{0} \\ e_{i-1} &= f_i^0 \cdot g_i^1 + f_i^1 \cdot g_i^0 + \forall v_i . e_i \end{aligned}$$

For our example, the elimination conditions are $E = [\mathbf{0}, \neg v_2, \mathbf{0}]$.

Before computing the intersection, we should normalize the operand sets by propagating the constraints imposed by the elimination conditions to remove the conflict inducing choices. In the example, we would get the normalized sets $F_n = [\mathbf{0}, \mathbf{1}, \mathbf{0}]$ and $G_n = [\mathbf{0}, \mathbf{1}, v_3]$ by substituting $\mathbf{1}$ for v_2 to eliminate vectors with the second bit 0. However, we can compute an approximation to the intersection with the original sets and then make a final (forward) pass to propagate the elimination constraints. The approximation K is obtained by:

$$\begin{aligned} k_i^1 &= f_i^1 + g_i^1 + e_{i|v_i=0} \\ k_i^0 &= f_i^0 + g_i^0 + e_{i|v_i=1} \end{aligned}$$

The correct intersection is obtained by substituting the restricted choices for the choice variables:

$$\begin{aligned} h_1^1 &= k_1^1 \\ h_1^0 &= k_1^0 \\ h_{i+1}^1 &= k_{i+1}^1|_{v_j \leftarrow h_j, 1 \le j \le i} \\ h_{i+1}^0 &= k_{i+1}^0|_{v_j \leftarrow h_j, 1 \le j \le i} \end{aligned}$$

The intersection algorithm requires a quadratic number of BDD operations. In symbolic reachability analysis (Figure 2), however, we use symbolic simulation for image computation and thus avoid intersection as part of the relational cross product.

2.5 Cofactors and Quantification

We can compute the Shannon-cofactors of vector F by cofactoring the individual components, i.e.,

$$F_{|x=c} = [f_{1|x=c}, \ldots, f_{n|x=c}]$$

where c is either 0 or 1. This corresponds to fixing a value for a choice variable.

Existential and universal quantification (set smoothing and consensus) can then be computed by the union and intersection of the cofactors with respect to the variable being quantified [1]

$$\begin{aligned} \exists x.F &= F_{|x=0} \cup F_{|x=1} \\ \forall x.F &= F_{|x=0} \cap F_{|x=1} \end{aligned}$$

[1] This is the same expansion as the domain partitioning in [6, 7]. However, since we have a union algorithm, we do not necessarily have to split recursively.

In reachability analysis (Figure 2), we will existentially quantify out the variables corresponding to the inputs and the choice variables for the current state elements as part of the re-parameterization procedure described below.

2.6 Re-Parameterization

Starting with a canonical representation of the inputs, we can obtain a Boolean functional vector for the outputs of a circuit by symbolic simulation. This vector, which represents each output value as a function of the input choice variables, must be *re-parameterized* to obtain a canonical representation for the output.

The re-parameterization is done by existentially quantifying the input choice variables. In general, any non-canonical Boolean functional vector can be made canonical by this procedure, by quantifying out the variables used in the non-canonical form.

2.7 Related Work

In [10], McMillan considers a canonical conjunctive decomposition of characteristic functions. We now show that this decomposition is closely related to the Boolean functional vector.

Given the choice variables $V = (v_1, \ldots, v_n)$ and the (canonical) Boolean functional vector $F = [f_1, \ldots, f_n]$, the vector $\tilde{F} = [v_1 \leftrightarrow f_1, \ldots, v_n \leftrightarrow f_n]$ represents a conjunctive decomposition of the characteristic function for the set, i.e., $\chi_{IMG(F)} = \bigwedge_{i=1}^{n} (v_i \leftrightarrow f_i)$. The difference between F and $\tilde{F}$ is that F maps an input vector to a vector in the represented set while $\tilde{F}$ represents a vector of constraints, one for each bit, which must be satisfied for set membership. Thus, $f_i = f_i^1 + f_i^c v_i$ evaluates the i-th bit while $\tilde{f}_i = f_i^1 v_i + f_i^0 \neg v_i + f_i^c$ is a constraint for the i-th bit.

The paper also gives algorithms for set manipulation using the conjunctive decomposition which, given the connection above, are in essence performing the same operations as our algorithms. The algorithms for the conjunctive decomposition are based on the generalized cofactor operation [6]. When the component order and the BDD variable order are the same, the BDD constrain operator can be used for the generalized cofactor. In this case, the algorithms with the conjunctive decomposition require fewer BDD operations than with Boolean functional vectors and are, hence, more efficient.

In [14], the authors present an algorithm to convert a non-canonical Boolean functional vector into a canonical form. They do so by maintaining a relation between the non-canonical and canonical expressions for each component. They also mention a possible optimization. However, without more details, it is hard to compare their algorithm with our re-parameterization procedure.

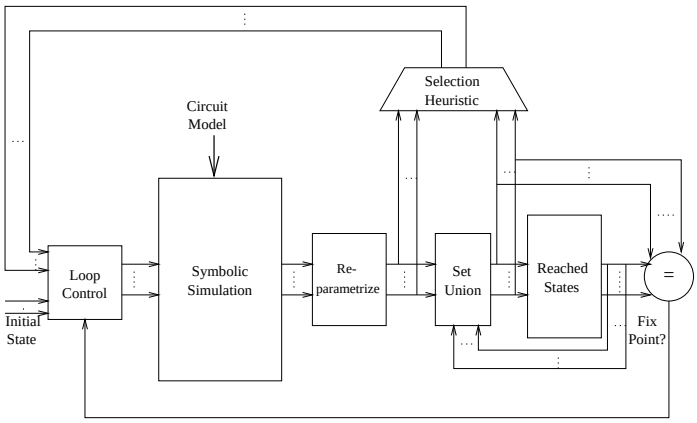

Figure 2. Reachability Analysis with Boolean Functional Vectors. Our algorithms make it unnecessary to convert to characteristic functions for set manipulations, as is done in Figure 1.

3 Symbolic Reachability Analysis with Boolean Functional Vectors

Our approach to symbolic reachability analysis is shown in Figure 2. Unlike Figure 1, we do not convert to the characteristic function at any stage. Instead, the re-parameterization and set union are performed on the Boolean functional vectors using either our algorithms or by converting to the conjunctive decomposition.

In our experiments described below, we used fixed variable orderings. The reason is that in addition to BDD variable ordering, we need to develop component reordering for the components of the Boolean functional vector. We used the same order for component ordering and BDD variable ordering. In this case, it is more efficient to use the algorithms from [10], as explained in Section 2.7.

We performed reachability experiments on some of the non-trivial ISCAS89 benchmark circuits. Table 2 shows the results obtained and compares them with the reachability analysis implemented in VIS [2] , using the IWLS95 set of heuristics [12] with default settings. The results for our approach are listed under BFV. Each row in the table lists the name of the circuit, the variable ordering used, the runtime in seconds and the peak live BDD node count in thousands. The variable orders we used[2] were the static ordering obtained from VIS (S1), the static ordering obtained from our tool (S2), an ordering obtained after a VIS run with dynamic ordering enabled (D), orders from the pdtexp distribution [5] (P) and others (O) available to us[3]. The experiments were run on an UltraSPARC-II 336MHz with the memory limit set to 1GB and the time limit set to 10 hours.

From the table we see that the new approach performs well with s3271 and s4863 but cannot complete s3330. On the other hand, we were able to complete s3330 with VIS, but not s3271. For s1512, BFV was much slower than VIS.

The two approaches differ in several aspects, e.g., we use a dynamic quantification schedule based on a simple support based cost heuristic. (Computing the cost dynamically does not impose much additional overhead, since we compute supports to avoid BDD operations on vector components that do not depend on the variable being quantified). The variable ordering requirements for the two representations can be quite different. For characteristic functions, it is essential that related variables occur together in the variable order. With Boolean functional vectors, the requirement is mainly that the important variables occur early in the order. Functional dependencies [9] are automatically factored out by the representation. Consider the characteristic function $\chi = (v_1 \leftrightarrow v_2) \wedge (v_3 \leftrightarrow v_4) \wedge (v_5 \leftrightarrow v_6)$ where a good variable ordering is one in which the pairs (v_1, v_2), (v_3, v_4) and (v_5, v_6) occur together. With the Boolean functional vector, all orderings are good in this case. We believe this property makes the variable ordering requirements less restrictive for Boolean functional vectors, e.g., in Table 2, BFV is able to complete the reachability of three of the benchmarks using a statically generated variable ordering (S1 or S2).

It is often the case that the Boolean functional vector representation is much smaller than the corresponding characteristic function. Table 3 lists the sizes of the characteristic function and the Boolean functional vector for the reachable states of s4863 (the size of the characteristic function BDD was obtained by converting the Boolean functional vector to the corresponding characteristic function). The size for BFV is the shared size of all the components; the individual components are usually much smaller, which can help avoid the intermediate blowup, e.g. in s4863 with order O, since

[2] We only list those cases where at least one of the two tools completed
[3] D and P are biased in favor of characteristic functions

		VIS - IWLS		BFV	
Name	Order	time(s)	Peak(K)	time(s)	Peak(K)
s1269	D	600	5958	1185	5691
	P	2135	14221	8656	29637
	O	8273	9384	592	2210
s1512	S2	11393	2551	18997	784
	D	990	1509	T.O.	
	O	13872	1889	18963	784
s3271	S2	T.O.		1531	2196
	D	T.O.		28414	16673
	P	T.O.		1617	1578
s3330	P	8866	3415		M.O.
	O	4797	19120	T.O.	
s4863	S1	T.O.		30	220
	D	51	2413	4481	17314
	P	1108	2514	17	175
	O		M.O.	19	102

Table 2. Results for Reachability Analysis with fixed variable orders for some ISCAS89 circuits using VIS and with Boolean functional vectors. T.O.(M.O.) indicates that the time (memory) limit was exceeded.

Order	S1	D	P	O
Char.Fn.	1455093	3387	8241	8515
BFV	19686	9817	6864	8851

Table 3. Sizes of the BDD for the characteristic function and the shared size of the BDDs for the Boolean functional vector for the reachable sets of s4863

.

the algorithms work on one component at a time.

In summary, the Boolean functional vector approach forms a viable alternative to the traditional reachability analysis method. The property of Boolean functional vectors to factor out functional dependencies can often reduce the variable ordering requirements.

4 Conclusions

We have presented algorithms for set union, intersection and quantification that work directly on a canonical Boolean functional vector representation. These algorithms enable efficient symbolic simulation based state-traversal. Our experiments indicate that in many cases, the Boolean functional vector representation is much more compact than the corresponding characteristic function. This results in significant performance gains.

In future work, we would like to develop a component reordering technique for components of the functional vector. We would also like to develop a symbolic simulation based model checker.

We would like to thank Fabio Somenzi and the anonymous reviewers for valuable comments.

References

[1] M. D. Aagaard, R. B. Jones, and C.-J. H. Seger. Formal Verification Using Parametric Representations of Boolean Constraints. In *Proceedings of the 36th Design Automation Conference (DAC' 99)*, pages 402–407, June 1999.

[2] R. K. Brayton, et al. VIS: a System for Verification and Synthesis. In *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, pages 428–432, New Brunswick, NJ, USA, July/Aug. 1996.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[4] R. E. Bryant and C.-J. H. Seger. Digital Circuit Verification using Partially-Ordered State Models. In *International Symposium on Multi-Valued Logic*, pages 2–7, May 1994.

[5] http://staff.polito.it/{˜cabodi,˜quer}.

[6] O. Coudert, C. Berthet, and J. C. Madre. Verification of Synchronous Sequential Machines Based on Symbolic Execution. In *Workshop on Automatic Verification Methods for Finite State Systems*, pages 365–373, 1989.

[7] O. Coudert and J. C. Madre. A Unified Framework for the Formal Verification of Sequential Circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 126–129, 1990.

[8] R. Hojati, S. Krishnan, and R. Brayton. Early quantification and partitioned transition relations. In *Proceedings of the IEEE International Conference on Computer Design(ICCD'96)* , pages 12–19, Oct. 1996.

[9] A. J. Hu and D. L. Dill. Reducing BDD size by exploiting functional dependencies. In *Proceedings of the 30th Design Automation Conference (DAC'93)*, pages 266–271, June 1993.

[10] K. L. McMillan. A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking. In *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV'96)*, pages 13–24, 1996.

[11] I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To Split or to Conjoin: the Question in Image Computation. In *Proceedings of the 37th Design Automation Conference (DAC'00)*, pages 23–28, 2000.

[12] R. Ranjan, A. Aziz, R. Brayton, B. Plessier, and C. Pixley. Efficient BDD Algorithms for FSM Synthesis and Verification. In *Workshop Notes of Intl. Workshop on Logic Synthesis (IWLS'95)*, May 1995.

[13] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit State Enumeration of Finite State Machines Using BDDs. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD'90)*, pages 130–133, Santa Clara, CA, Nov. 1990.

[14] J. Yang and C.-J. H. Seger. Generalized Symbolic Trajectory Evaluation - Abstraction in Action. In *Formal Methods in Computer-Aided Design (FMCAD'02)*, Nov. 2002.